求解 0-1 背包问题的混合蝙蝠算法 *

万晓琼，张惠珍†

(上海理工大学 管理学院，上海 200093)

摘　要：针对基本蝙蝠算法易陷入局部最优、收敛速度慢等缺点，对其进行优化研究。基于 0-1 背包问题的具体特征，在基本蝙蝠算法原有概念和框架的基础上，引入遗传算法中的交叉机制以及反置算子建立全新的位置转移方式和局部搜索规则；加入贪心策略进行解的可行化和充分利用，增强局部搜索能力，加快算法收敛速度，构建全新的混合蝙蝠算法。将混合蝙蝠算法应用于两组 0-1 背包算例，仿真实验结果优于自适应元胞粒子群算法、基本蝙蝠算法和贪心二进制蝙蝠算法。结果验证了该混合算法求解 0-1 背包问题的可行性和有效性。

关键词：0-1 背包问题；蝙蝠算法；遗传算法；反置算子；贪心策略

中图分类号：TP301.6　　doi: 10.3969/j.issn.1001-3695.2018.01.0137

Hybrid bat algorithm for solving 0-1 knapsack problem

Wan Xiaoqiong, Zhang Huizhen†

(*School of Management*, *University of Shanghai for Science & Technology*, *Shanghai* 200093, *China*)

Abstract: In order to overcome the shortcomings of the basic bat algorithm, such as easy to fall into local optimum and slow convergence speed, the optimization study was carried out. This paper proposed a new hybrid bat algorithm based on the original features, framework of the basic bat algorithm and specific characteristics of the 0-1 knapsack problem. The hybrid algorithm introduced the crossover mechanism of the genetic algorithm and the inverse operator to construct a brand-new location transfer method and local search rule. The greedy strategy was added to make the solution feasible and fully utilized, enhancing the local search ability and speeding up the convergence. This hybrid bat algorithm was applied to two sets of 0-1 knapsack cases. The simulation results exceeded adaptive cellular particle swarm optimization, basic bat algorithm and greedy binary bat algorithm. The results verify the feasibility and effectiveness of the hybrid algorithm in solving 0-1 knapsack problem.

Key words: 0-1 knapsack problem; bat algorithm; genetic algorithm; inverse operator; greedy strategy

0　引言

背包问题(knapsack problem, KP) 是运筹学中一个典型的 NP-Complete 组合优化问题，旨在寻求满足背包约束的条件下具有最大价值的物品装载方案，现实生活中的诸多问题都可被归结为背包算例，如投资决策问题、任务调度问题、货物装载问题、材料切割问题等。背包问题一般可以分为三类：0-1 背包问题（0-1 knapsack problem, 0-1 KP）[1]、完全背包问题和多重背包问题，其中 0-1 背包问题是最基本的背包问题，包含了背包问题设计状态以及方程的最基本思想，其他类型的背包问题也可以转化为 0-1 背包问题进行求解。本文旨在对 0-1 背包问题进行研究。

0-1 背包问题自提出以来，一直是众多学者研究的热点。从已有的研究成果可以看出，目前求解该问题的算法主要可以分为两类：精确算法和启发式算法。精确算法如：分支定界法[2,3]、动态规划法[4]、回溯法[5]、贪心算法[6]等。精确算法虽然可以求得优化问题的最优解，但无法解决随着物品数量的增加而产生的“组合爆炸”问题，启发式算法的出现，为解决此类问题开辟了一条新途径。现在已有多种启发式算法被用来解决 0-1 背包问题，如蚁群算法[7,8]、和声算法[9~11]、遗传算法[12,13]、粒子群算法[14,15]、狼群算法[16]、鲸鱼算法[17]、蜻蜓算法[18]等。上述算法在解决 0-1 背包问题时取得了一定的成果，同时也显现出易早熟、后期收敛速度慢等不足。为了克服单一启发式算法的缺点，运用混合算法求解组合优化问题成为趋势，如混合遗传算法[19]、基于遗传算子的蚁群算法[20]、基于贪心策略的遗传算法[21]等。

蝙蝠算法（bat algorithm, BA）[22]是剑桥大学学者 Yang Xinshe 基于蝙蝠回声定位行为于 2010 年提出的一种启发式算法，具有参数少、结构简单、鲁棒性强等优点。然而，与其他

收稿日期：2018-01-27；**修回日期：**2018-04-12　**基金项目：**国家自然科学基金资助项目（71401106）；国家教育部人文社科规划基金资助项目（16YJA630037）

作者简介：万晓琼（1992-），女，河南洛阳人，硕士研究生，主要研究方向为智能优化；张惠珍（1979-），女（通信作者），副教授，硕导，博士，主要研究方向为运筹学、智能优化（zhzzywz@163.com）.

启发式算法相似，基本蝙蝠算法也存在易陷入局部最优和收敛速度慢等缺点。针对这些缺点，本文在基本蝙蝠算法原有概念和结构的基础上，引入遗传算法中的交叉机制提高全局搜索能力，加入反置算子和贪心策略增强局部搜索能力，设计出全新的混合蝙蝠算法（hybrid bat algorithm, HBA）来求解 0-1 背包问题，并与文献[23,24]的 0-1 背包算例相对比，结果验证了该算法在寻优能力和收敛速度上均有较大优势。

1　0-1 背包问题

0-1 背包问题是经典的组合优化问题，具体可以描述为：给定有重量限制的背包以及具有重量和价值的 n 个待装物品，求解目标为在背包载重限制下选择物品装入背包并使得装入物品价值最大。该问题数学模型表示如下：

$$max \quad F(x)=\sum_{i=1}^{n} v_i x_i \tag{1}$$

$$s.t. \quad \sum_{i=1}^{n} w_i x_i \le C \tag{2}$$

$$x_i=\begin{cases}1，物品\ i\ 装入背包中\\ 0，\quad 否则\end{cases} \quad \forall\, i\in I \tag{3}$$

模型中，$I=\{i \mid i=1,2,\ \ldots n\}$ 为物品集；$v=\{v_i \mid i=1,2,\ \ldots n\}$ 为物品价值集；$w=\{w_i \mid i=1,2,\ \ldots n\}$ 为物品重量集；C为背包载重量；目标函数式（1）表示最大化装入背包物品价值；约束式（2）保证装入背包物品的总重量不超出背包载重量；约束式（3）中 x_i 为决策变量。

2　混合蝙蝠算法

蝙蝠算法是自然界中蝙蝠群体回声定位行为的智能模拟，是一种搜索全局近似解的智能优化算法。蝙蝠个体是蝙蝠算法的基本单元，它们通过嘴巴和鼻孔向外发射很强的超声波，并通过超凡的大耳廓接收回音，再依据发出超声波和接收到回声之间的时差、到达两耳之间的强度差、回声定位波形变化建立空间三维场景来感知猎物或障碍物的距离、方位以及性质，进而制定捕捉猎物和躲避障碍物的计划。算法思想可以概括为：蝙蝠种群随机散布于n 维问题空间内，把它们映射为优化问题空间内的 s 个解，所优化问题的目标函数对应解的适应度值，适应度值决定解的优劣，算法的优化过程即为蝙蝠种群的其他个体调整响度、脉冲发射率在解空间中追随最优蝙蝠的迭代过程。

2.1　求解 0-1 背包问题的混合蝙蝠算法

蝙蝠算法最初被用来求解函数优化问题，近几年有学者提供新的改进思想和方法去解决离散问题。在本文中，为了把蝙蝠算法应用于 0-1 背包问题，在求解优化问题的过程中，利用二进制编码方式初始化蝙蝠位置，在蝙蝠位置更新规则中引入遗传算法的交叉机制，利用反置算子构建全新的局部搜索规则，称之为混合蝙蝠算法，下文从初始化蝙蝠位置、蝙蝠速度与位置更新、局部搜索以及音量和脉冲的更新四个方面来介绍混合蝙蝠算法。

2.1.1 *初始化蝙蝠位置规则*

根据 0-1 背包问题解的性质，本文采用二进制编码方式随机初始化蝙蝠种群位置。用$1*n$维向量 $x_i=(x_{i1},\ x_{i2},\ldots,x_{in})$ 表示第 i 只蝙蝠的初始位置，其中 $x_{ij}\in\{0,1\}$，$1\le j\le n$，当 $x_{ij}=1$ 时，物品 j 装入背包，反之，物品 j 不装入背包。具体映射关系如式（4）所示。

$$x_{ij}=\begin{cases}1，\quad rand>0.5\\ 0，\quad rand\le 0.5\end{cases} \tag{4}$$

2.1.2 *蝙蝠速度与位置更新规则*

1）速度更新规则

由于离散优化问题的特殊性，在初始化蝙蝠位置以及后续速度定义、位置转移编码都是一步到位的，故删去基本蝙蝠算法中频率 f 这个参数。重新定义第 i 只蝙蝠 x_i 和最优蝙蝠 x_* 之间的 Manhattan 距离为蝙蝠的速度。则第 i 只蝙蝠在定义的搜索空间中第 t 时刻v_i^t的更新公式为

$$v_i^t=\sum_{j=1}^{n}\left|x_{ij}-x_{*j}\right| \tag{5}$$

2）位置更新规则

本文采用遗传算法中的交叉机制进行位置转移，一只蝙蝠对应一条染色体，位置更新过程主要可以分为两步：

a)子代遗传父代的相同基因。具体为：设最优蝙蝠x_*为父代 1，第 i 只蝙蝠 x_i为父代 2，对比父代 1、2 相应的基因位，当基因相同（即相应基因位上同为 1 或 0）时，将相应基因直接遗传给子代；当基因不同时，子代相应基因位以通配符 * 填充，记此时的子代蝙蝠位置为x_{mid}。

b) x_{mid}中通配符的个数为v_i^t，即为第 i 只蝙蝠的转移速度，对于这v_i^t个不同的基因，设置追随概率p_{fol}，建立转移公式如式（6）所示。

$$x_{new}=x_{mid}\left(v_i^t,\ p_{fol}\right) \tag{6}$$

具体为：当$rand>p_{fol}$时，子代追随最优蝙蝠，遗传父代 1 的对应基因，反之，遗传父代 2 的对应基因，最后更新子代蝙蝠位置为 x_{new}。利用这种交叉机制，一方面子代可以遗传父代优秀的基因。另一方面，加入追随概率p_{fol}，增添跟从最优蝙蝠的随机性以维持蝙蝠种群的多样化，一定程度上克服了基本蝙蝠算法容易早熟的缺点。以$n=6$为例，具体交叉重组过程如图 1 所示。

2.1.3 *局部搜索规则*

本文采用的局部搜索方式有两种，一为利用反置算子对最优蝙蝠进行局部搜索；二为使用贪心策略对不可行解和非充分利用解进行局部搜索。

1）反置算子

相对于其他智能优化算法，蝙蝠算法可以通过比较脉冲发生率 r_i 和随机数的大小，实现对全局搜索和局部搜索之间动态转换。即当 $rand>r_i$ 时，在当前最优解集中选取一个最优蝙蝠，进行随机游走，产生局部新解。由于二进制编码的特殊性，基

本蝙蝠算法的局部搜索方式并不能取得优异结果。本文引入反置算子设置全新的局部搜索规则，具体为：反置概率 p_{not} 随机选取最优蝙蝠的 $n*p_{not}$ 个基因进行反置处理，最后更新局部新解 x_{new}。这种随机反置的局部搜索方式使得随着迭代次数增加而趋于相似的蝙蝠种群容易跳出局部最优。对于选定的基因位 $j_{not}\in J_{not}$ 具体优化方法如下：

$$x_{new j_{not}}=\begin{cases}1, & x_{*j_{not}}=0\\ 0, & x_{*j_{not}}=1\end{cases} \tag{7}$$

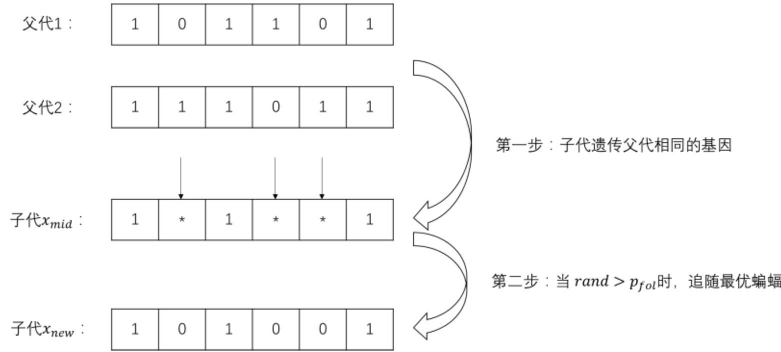


图 1 蝙蝠算法位置更新过程

2）贪心策略

a）对不可行解进行贪心策略。0-1 背包问题作为具有约束条件的组合优化问题，在蝙蝠位置随机初始化、位置更新以及反置算子局部搜索部分均有可能产生无效解，即装入物品总重量超出背包重量限制，针对这些不可行解若直接舍弃，则会丢失已搜索到的信息，导致算法收敛速度变慢，故本文引入贪心策略对不可行解进行局部搜索：计算已装入背包物品的价值比 $vol_i=v_i/w_i$，依次舍去低价值比的物品，直至剩余物品总重量不超出背包载重。

b）对可行解进行贪心策略。由于初始化解的随机性和可行化解的限制，可能导致背包容量的不充分利用，针对这部分解采用贪心策略进行局部搜索：计算未装入背包的物品价值比 $vol_i=v_i/w_i$，依次加入高价值比的物品，直至背包利用率达到最大。

2.1.4 蝙蝠音量和脉冲发生率更新规则

在蝙蝠接近猎物时，超声波音量逐渐降低，脉冲发生率逐步提高。音量A_i^t和脉冲发生率r_i^t的更新公式为

$$A_i^t=\alpha A_i^{t-1} \tag{8}$$

$$r_i^t=r_i^0\,\{1-exp[-\gamma(t-1)]\} \tag{9}$$

其中：α 为音量衰减系数，γ为脉冲发生率增长系数，r_i^0为最大脉冲频率。对任意$0<\alpha<1$，$\gamma>0$，可以看出：$A_i^t\to 0$，$r_i^t\to r_i^0$， $as\quad t\to\infty$。

2.2 求解 0-1 背包问题的混合蝙蝠算法设计

通过对基本蝙蝠算法的改造，设计出全新的求解 0-1 背包问题的混合蝙蝠算法。具体步骤如下：

a)初始化蝙蝠种群。蝙蝠种群大小为 s，第 i 只蝙蝠的位置为x_i，速度为 v_i，追随概率p_{fol}，反置概率p_{not}，音量为 A_i^0，脉冲发生率为 r_i^0，最大迭代次数N_gen，并根据式（4）初始化蝙蝠位置。

b)蝙蝠速度和位置更新。计算得出最优蝙蝠x_*，运用式（5）更新蝙蝠速度v_i^t，利用遗传算法的交叉机制更新蝙蝠位置x_{new}。

c)局部搜索。若 $rand>r_i$， 根据式（7）对最优解集中选取的最优解进行局部搜索，得出局部新解x_{new}。

d)更新满意解。若 $rand<A_i$ 且 $F(x_{new})<F(x_*)$，更新当前满意解及其对应的目标函数值。

e)更新音量和脉冲发生率。通过式（8）减小 A_i，式（9）增大 r_i。

f)排列蝙蝠并找到当前最优蝙蝠 x_*。

g)$N_iter = N_iter+1$，若$N_iter > N_gen$，则结束搜索并输出结果，否则转 b)。

（注：在初始化蝙蝠种群、蝙蝠位置更新和反置算子局部搜索后均要使用贪心策略进行解的可行化和充分利用。）

3 仿真实验与分析

为了验证本文提出的混合蝙蝠算法的有效性，文章选用两组算例测试其求解性能。第 1 组算例选用文献[23]的 9 个 0-1 背包问题进行测试，并将其求解结果与自适应元胞粒子群算法(adaptive cellular particle swarm optimization, ACPSO)[23]的测试结果进行对比分析。第 2 组算例选用文献[24]的三个 0-1 背包问题进行测试，并与基本蝙蝠算法以及贪心二进制蝙蝠算法（greedy binary bat algorithm，GBBA）[24]进行比较分析。

本文仿真实验所用硬件环境为 Intel(R) Core™ i5-3320 CPU @2.60GHz，2.0 GB 内存，64 位 Windows 8 操作系统，编程语言为 MATLAB R2015a。文献[23]自适应元胞粒子群算法的仿真计算环境为 Core™ 2 Duo CPU 为 2.93 GHz，内存 1.93 GB，操作系统 Windows XP，编程语言为 MATLAB。文献[24] 贪心二进制蝙蝠算法的仿真实验所用的硬件环境为 Intel ® Core ™ Duo CPU T2400 1.83 GHz，内存为 1 GB，操作系统为 Windows XP2002，编程语言为 VC++ 6.0。

为体现算法对比结果的公平性，本文求解第 1 组算例的蝙蝠种群大小s、最大迭代次数N_gen、单个算例运行次数 T 和文献[23]的粒子群算法中的种群大小、最大迭代次数及单个算例运行次数设置相同，即$s=50$，$N_gen=500$，T = 30。其他参

数遵从算法的具体特征，设置为：初始音量$A_i^0 = 0.25$，初始脉冲发生率 $r_i^0 = 0.5$，$\alpha = \gamma = 0.9$，追随概率 $p_{fol} = 0.5$，反置概率 $p_{not} = 0.2$。

表 1 给出第 1 组 9 个测试算例的维数、物品重量集w、物品价值集v、背包载重量 C 以及已知最优值。

表 1　0-1 背包问题的 9 组仿真算例

算例	维数	背包问题	已知最优值
KP1	10	w = (95,4,60,32,23,72,80,62,65,46); v = (55,10,47,5,4,50,8,61,85,87); C =269	295
KP2	20	w=(92,4,43,83,84,68,92,82,6,44,32,18,56,83,25,96,70,48,14,58);v= [44,46,90,72,91,40,75,35,8,54,78,40,77,15,61,17,75,29,75,63]; C =878	1024
KP3	20	w=(44,46,90,72,91,40,75,35,8,54,78,40,77,15,61,17,75,29,75,63);v=(92,4,43,83,84,68,92,82,6,44,32,18,56,83,25,96,70,48,14,58); C =878	1042
KP4	50	w =(95,39,69,63,49,104,56,58,47,23,17,129,91,28,77,125,73,5,103,63,76,23,47,79,119,125,26,119,79,56,50,75,12,26,31,43,41,38,29,21,14,9,3,17,8,8,9,7,4,5); v =(293,291,290,280,278,274,269,265,248,247,245,245,241,234,229,228,222,216,214,191,191,187,171,170,164,152,142,132,131,126,122,116,112,111,110,106,77,76,74,73,69,67,42,41,35,33,30,29,28,26); C =959	4882
KP5	100	w = (54,95,36,18,4,71,83,16,27,84,88,45,94,64,14,80,4,23,75,36,90,20,77,32,58,6,14,86,84,59,71,21,30,22,96,49,81,48,37,28,6,84,19,55,88,38,51,52,79,55,70,53,64,99,61,86,1,64,32,60,42,45,34,22,49,37,33,1,78,43,85,24,96,32,99,57,23,8,10,74,59,89,95,40,46,65,6,89,84,83,6,19,45,59,26,13,8,26,5,9); v = (297,295,293,292,291,289,284,284,283,283,281,280,279,277,276,275,273,264,260,257,250,236,236,235,235,233,232,232,228,218,217,214,211,208,205,204,203,201,196,194,193,193,192,191,190,187,187,184,184,184,181,179,176,173,172,171,160,128,123,114,113,107,105,101,100,100,99,98,97,94,94,93,91,80,74,73,72,63,63,62,61,60,56,53,52,50,48,46,40,40,35,28,22,22,18,15,12,11,6,5); C = 3820	15170
KP6	100	w=(54,183,106,82,30,58,71,166,117,190,90,191,205,128,110,89,63,6,140,86,30,91,156,31,70,199,142,98,178,16,140,31,24,197,101,73,169,73,92,159,71,102,144,151,27,131,209,164,177,177,129,146,17,53,164,146,43,170,180,171,130,183,5,113,207,57,13,163,20,63,12,24,9,42,6,109,170,108,46,69,43,175,81,5,34,146,148,114,160,174,156,82,47,126,102,83,58,34,21,14);v=(597,596,593,586,581,568,567,560,549,548,547,529,529,527,520,491,482,478,475,475,466,462,459,458,454,451,449,443,442,421,410,409,395,394,390,377,375,366,361,347,334,322,315,313,311,309,296,295,294,289,285,279,277,276,272,248,246,245,238,237,232,231,230,225,192,184,183,176,174,171,169,165,165,154,153,150,149,147,143,140,138,134,132,127,124,123,114,111,104,89,74,63,62,58,55,48,27,22,12,6); C =6718	26559
KP7	100	w=(94,81,8,62,21,83,85,45,48,81,4,64,59,97,96,14,21,59,34,68,36,2,3,65,26,48,25,17,11,97,43,23,24,48,56,73,54,15,98,99,47,93,78,68,24,52,8,89,100,7,9,46,8,40,77,46,76,78,7,32,92,11,93,75,6,60,64,15,99,30,99,61,17,3,31,34,76,68,79,91,95,25,73,43,89,9,12,31,71,24,19,70,76,14,50,85,40,78,12,6);v=(94,60,88,18,39,57,4,74,86,77,59,45,74,99,46,68,99,83,23,85,80,41,58,11,35,73,100,2,79,58,70,40,6,9,26,2,7,92,40,45,65,50,80,53,37,84,14,14,72,41,46,76,13,78,77,77,49,29,63,61,32,2,6,47,31,46,35,85,39,64,52,24,25,26,50,81,89,61,44,95,40,27,83,81,85,32,60,91,44,54,31,48,50,94,32,83,24,40,43,11); C =999.60	2660
KP8	100	w=(94,81,8,62,21,83,85,45,48,81,4,64,59,97,96,14,21,59,34,68,36,2,3,65,26,48,25,17,11,97,43,23,24,48,56,73,54,15,98,99,47,93,78,68,24,52,8,89,100,7,9,46,8,40,77,46,76,78,7,32,92,11,93,75,6,60,64,15,99,30,99,61,17,3,31,34,76,68,79,91,95,25,73,43,89,9,12,31,71,24,19,70,76,14,50,85,40,78,12,6);v=(94,60,88,18,39,57,4,74,86,77,59,45,74,99,46,68,99,83,23,85,80,41,58,11,35,73,100,2,79,58,70,40,6,9,26,2,7,92,40,45,65,50,80,53,37,84,14,14,72,41,46,76,13,78,77,77,49,29,63,61,32,2,6,47,31,46,35,85,39,64,52,24,25,26,50,81,89,61,44,95,40,27,83,81,85,32,60,91,44,54,31,48,50,94,32,83,24,40,43,11); C =2499	4143
KP9	100	w=(94,81,8,62,21,83,85,45,48,81,4,64,59,97,96,14,21,59,34,68,36,2,3,65,26,48,25,17,11,97,43,23,24,48,56,73,54,15,98,99,47,93,78,68,24,52,8,89,100,7,9,46,8,40,77,46,76,78,7,32,92,11,93,75,6,60,64,15,99,30,99,61,17,3,31,34,76,68,79,91,95,25,73,43,89,9,12,31,71,24,19,70,76,14,50,85,40,78,12,6);v=(94,60,88,18,39,57,4,74,86,77,59,45,74,99,46,68,99,83,23,85,80,41,58,11,35,73,100,2,79,58,70,40,6,9,26,2,7,92,40,45,65,50,80,53,37,84,14,14,72,41,46,76,13,78,77,77,49,29,63,61,32,2,6,47,31,46,35,85,39,64,52,24,25,26,50,81,89,61,44,95,40,27,83,81,85,32,60,91,44,54,31,48,50,94,32,83,24,40,43,11); C =3998.4	4986

每个算例独立测试 30 次，结果如表 2 所示。其中，实验最优值、实验平均值、实验最差值由 30 次运行结果对比得出；总消耗时间为 30 次独立实验的运行总时间；成功次数为 30 次测试中找到当前最优解的次数；最小迭代次数、平均迭代次数、最大迭代次数为找到最优值的前提下对比实验结果得出。表 2 中 ACPSO 算法的实验结果均来自文献[23]。

由表 2 可看出，对于算例 KP9，HBA 算法得出的实验最优解为 4987，实验平均值为 4986.5，优于 ACPSO 给出的已知最优值 4986，说明 HBA 的全局搜索能力优于 ACPSO；对于算例 KP1~KP8，从成功次数可以看出，HBA 运行的结果除算例 KP6 以外，其他算例 30 次测试均能达到最优值，而 ACPSO 只有 KP1、KP5、KP8 能 100%达到最优值，二者对比说明 HBA 寻优的鲁棒性较强；且从迭代次数的三个实验指标可以看出，HBA 能在较低的迭代次数中达到最优解，除 KP5 算例外，均远优于 ACPSO，说明 HBA 具有较快的收敛速度。综合 HBA 的成功次数以及寻优成功的迭代次数两个方面分析，HBA 消耗的总时间显然优于 ACPSO。综合所有的实验指标可以看出，HBA 的寻优能力远优于 ACPSO，在求解 0-1 背包问题中具有较强的求解性能。

表 2 实验结果对比

算例	算法	维数	当前最优值	实验最优值	实验平均值	实验最差值	消耗总时间/s	成功次数	最小迭代次数	平均迭代次数	最大迭代次数
KP1	ACPSO	10	295	295	295	295	0.19	30	1	1.80	6
	HBA			295	295	295	0.63	30	1	1.00	1
KP2	ACPSO	20	1024	1024	1.0212e +003	1018	50.44	16	1	194.19	441
	HBA			1024	1024	1024	0.62	30	1	1.43	4
KP3	ACPSO	20	1042	1042	1.0393e +003	1037	56.45	14	49	235.00	474
	HBA			1042	1042	1042	3.29	30	1	27.23	142
KP4	ACPSO	50	4882	4882	4. 8571e +003	4834	131.90	5	91	133.93	421
	HBA			4882	4882	4882	1.51	30	1	6.73	39
KP5	ACPSO	100	15170	15170	15170	15170	0.94	30	1	1.00	1
	HBA			15170	15170	15170	0.91	30	1	1.83	2
KP6	ACPSO	100	26559	26559	2.6527e +004	26525	367.52	1	45	45.00	45
	HBA			26559	2.6551e+004	26534	35.47	14	2	10.43	103
KP7	ACPSO	100	2660	2660	2660	2660	203.25	13	20	190.38	417
	HBA			2660	2660	2660	1.27	30	1	2.47	5
KP8	ACPSO	100	4143	4143	4143	4143	37.73	30	4	58.83	162
	HBA			4143	4143	4143	1.45	30	3	4.57	9
KP9	ACPSO	100	4986	4986	4986	4986	0.80	30	1	1.00	1
	HBA			4987	4.9865e+003	4986	31.17	15	2	4.73	33

为充分测试各算法的求解性能，体现对比公平性，本文在测试第 2 组算例时，混合蝙蝠算法中初始音量$A_i^0 = 0.005$、初始脉冲发生率 $r_i^0 = 0.75$、音量衰减系数$\alpha = 0.95$、脉冲发生率增长系数$\gamma = 0.7$、单个算例测试次数 $T = 50$ 均沿用文献[24]的设置，追随概率 p_{fol}、反置概率 p_{not}同第 1 组算例。第 2 组算例选用文献[24]的 3 个算例，其中算例 1、算例 2 和第 1 组算例中的 KP1、KP2 相同，具体见表 1；算例 3 如表 3 所示。

对 3 个算例分别进行测试，结果如表 4、表 5 所示。表 4 为 BA、GBBA 与 HBA 求解算例 1~3 达到最大迭代次数时的目标函数值的比较。对比 50 次测试结果得出最差值、平均值和最优值；表 5 为 BA、GBBA 与 HBA 求解算例 1~3 的收敛性比较，当算法在最大迭代次数限制内达到最优值时为寻优成功，记录下迭代次数，对比 50 次运行结果得出最小数、平均数和成功率。表 4、表 5 中 GBBA 的实验结果均来自文献[24]。对于算例 3，在目标函数值的对比中最大迭代次数设置为 300，在收敛比较中最大迭代次数设置为 500，为了实验对比结果的公平性，本文和文献[24]设置相同。

表 3　0-1 背包问题的仿真

算例	维数	背包问题	已知最优值
算例3	50	**w** = [438,754,699,587,789,912,819,347,511,287,541,784,676,198,572,914,988,4,355,569,144,272,531,556,741,489,321,84,194,483,205,607,399,747,118,651,806,9,607,121,370,999,494,743,967,718,397,589,193,369];**v**=[72,490,651,833,883,489,359,337,267,441,70,934,467,661,220,329,440,774,595,98,424,37,807,320,501,309,834,851,34,459,111,253,159,858,793,145,651,856,400,285,405,95,391,19,96,273,152,473,448,231];**C**=11258;	16102

表 4 BA、GBBA 与 HBA 求解算例 1~3 的目标函数值比较

算例	算法	种群规模	最大迭代次数	最差值	平均值	最优值
算例1	BA	4	300	158	226. 42	295
	GBBA			294	294. 80	295
	HBA			294	294. 90	295
算例2	BA	4	300	813	918. 36	995
	GBBA			1018	1019	1024
	HBA			1018	1023. 40	1024
算例3	BA	15	300	7989	10126. 12	12224
	GBBA			16012	16053	16102
	HBA			16029	16086. 84	16102

表 5 BA、GBBA 与 HBA 求解算例 1~3 的收敛性比较

算例	算法	种群规模	最大迭代次数	最小数	平均数	成功率
算例1	BA	15	300	4	5	3/50
	GBBA			5	5.3	47/50
	HBA			1	11.56	50/50
算例2	BA	15	300	4	4	1/50
	GBBA			8	19.29	41/50
	HBA			1	13.26	50/50
算例3	BA	15	500	0	0	0/50
	GBBA			10	33.56	18/50
	HBA			2	65.95	44/50

由表 4 的最优值可以看出 HBA 和 GBBA 均能求得算例 1~3 的最优解，而基本 BA 算法仅能求得算例 1 的最优解，表明改进 BA 算法一定程度上克服了 BA 易陷入局部最优的缺点，寻优能力均优于 BA。由表 4 的平均值可以看出 HBA 的求解结果均优于另外两种算法，且从表 5 的成功率可以看出，对于算例 1 和算例 2，HBA 算法均能 100%求得最优解，求解算例 3 的成功率也优于其他两种算法，综合两个实验指标可以看出 HBA 的平均求解精度和求解鲁棒性均优于 BA 和 GBBA。综合表 4、5 可以看出，HBA 求解 0-1 背包问题的综合能力优于 BA 和 GBBA。

为了更直观地展现出 HBA 的求解效果，图 2~4 给出 BA 和 HBA 求解算例 1~3 的迭代过程图。如图所示，HBA 能在较小的迭代次数内求得最优值，而 BA 在进行一定的迭代次数后就陷入了局部最优，由此可以看出 HBA 克服了 BA 易早熟以及收敛速度慢的缺点，全局搜索能力和收敛速度均远优于 BA。

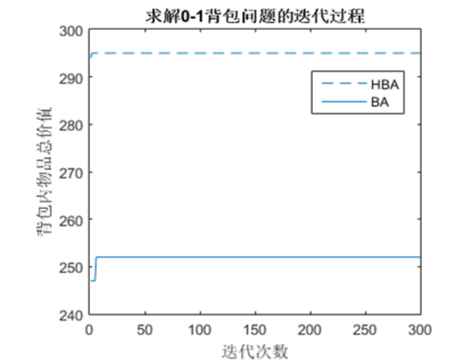


图 2 求解算例 1 的迭代过程图

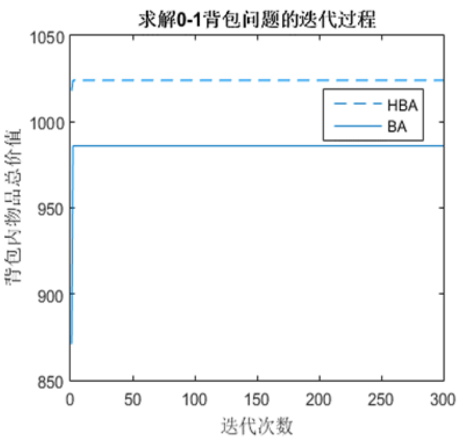


图 3 求解算例 2 的迭代过程图

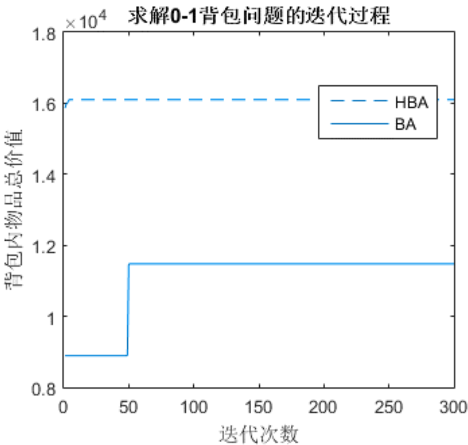


图 4 求解算例 3 的迭代过程图

4 结束语

本文针对 0-1 背包问题的具体特征，在基本蝙蝠算法的框架基础上加入遗传算法、反置算子和贪心策略，利用四者各自优点，实现优势互补，设计出一种全新的混合蝙蝠算法。该算法克服了基本蝙蝠算法易陷入局部最优和收敛速度慢的缺点。通过对 0-1 背包算例的仿真对比研究，本文提出的混合蝙蝠算法的综合求解能力优于自适应元胞粒子群算法、基本蝙蝠算法和贪心二进制蝙蝠算法，是求解 0-1 背包问题的一种有效方法。

参考文献：

[1] Fayard D, Plateau G. Resolution of the 0-1 knapsack problem comparison of methods [J]. Mathematical Programming, 1975, 8 (1): 272-307.

[2] 盛红波，孙娟，孙小玲. 0-1 多项式背包问题的一种精确算法 [J]. 上海大学学报：自然科学版, 2006, 12 (4): 389-393. (Sheng Hongbo, Sun Juan, Sun Xiaoling. A rigor method for solving 0-1 polynomial knapsack problem [J]. Joural of the Shanghai University: Natural Science, 2006, 12 (4): 389-393.)

[3] Shen Jingcheng, Shigeoka K, Ino F, *et al.* An out-of-core branch and bound method for solving the 0-1 knapsack problem on a GPU [J]. International Conference on Algorithms & Architectures for Parallel Processing, 2017: 254-267.

[4] Elkihel M, Baz D E. An efficient dynamic programming parallel algorithm for the 0-1 kanpsack problem [J]. Parallel Computing-advances & Current Issues-the International Conference Parco, 2015: 298-305.

[5] 徐颖. 回溯法在 0-1 背包问题中的应用 [J]. 软件导刊, 2008 (12): 54-55. (Xu Ying. Application of backtracking method in 0-1 knapsack problem [J]. Software Guide, 2008 (12): 54-55.)

[6] 史岚, 张义宏, 吕建辉. 基于绝对贪心和预期效率的 0-1 背包问题优化 [J]. 计算机应用研究, 2014, 31 (3): 684-687. (Shi Lan, Zhang Yihong, Lv Jianhui. Optimization algorithm of 0-1 knapsack problem based on absolute greedy and expected efficiency [J]. Application Research of Computers, 2014, 31 (3): 684-687.)

[7] 胡小兵, 黄席樾. 基于蚁群优化算法的 0-1 背包问题求解 [J]. 系统工程学报, 2005, 20 (5): 520-523. (Hu Xiaobing, Huang Xiyue. Solving 0-1 knapsack problem based on ant colony optimization algorithm [J]. Journal of Systems Engineering, 2005, 20 (5): 520-523.)

[8] 秦玲, 白云, 章春芳等. 解 0-1 背包问题的蚁群算法 [J]. 计算机工程, 2006, 32 (6): 212-214. (Qin Ling, Bai Yun, Zhang Chunfang, *et al.* Ant colony algorithm for 0-1 knapsack problem [J]. Computer Engineering, 2006, 32 (6): 212-214.)

[9] 李若平, 欧阳海滨, 高立群, 等. 学习型和声搜索算法及其在 0-1 背包问题中的应用 [J]. 控制与决策, 2013, 28 (2): 205-210. (Li Ruoping, Ouyang Haibin, Gao Liqun, *et al.* Learned harmony search algorithm and its application to 0-1 knapsack problems [J]. Control and Decision, 2013, 28 (2): 205-210.)

[10] Wang Ling, Yang Ruixin, Xu Yin, *et al.* An improved adaptive binary Harmony Search algorithm [J]. Information Sciences, 2013, 232 (5): 58-87.

[11] Zou Dexuan, Gao Liqun, Li S, *et al.* Solving 0–1 knapsack problem by a novel global harmony search algorithm [J]. Applied Soft Computing Journal, 2011, 11 (2): 1556-1564.

[12] 王莉, 绍定宏, 陆金桂. 基于遗传算法的 0//1 背包问题求解 [J]. 计算机仿真, 2006, 23 (3): 154-156. (Wang Li, Shao Dinghong, Lu Jingui. The genetic algorithm of solving 0//1 knapsack problem [J]. Computer Simulation, 2006, 23 (3): 154-156.)

[13] Caro F. A flipping local search genetic algorithm for the multidimensional 0-1 knapsack problem [C]// Proc of Spanish Association Conference on Current Topics in Artificial Intelligence. 2005: 21-30.

[14] Zhang Guoli, Wei Yi. An improved particle swarm optimization algorithm for solving 0-1 knapsack problem [J]. International Conference on Machine Learning & Cybernetics, 2008, 2: 915-918.

[15] Haddar B, Khemakhem M, Rhimi H. A quantum particle swarm optimization for the 0–1 generalized knapsack sharing problem [J]. Natural Computing, 2016, 15 (1): 153-164.

[16] 吴虎胜, 张凤鸣, 战仁军, 等. 求解 0-1 背包问题的二进制狼群算法 [J]. 系统工程与电子技术, 2014, 36 (8): 1660-1667. (Wu Husheng, Zhang Fengming, Zhan Renjun, *et al.* A binary wolf pack algorithm for solving 0-1 knapsack problem [J]. Systems Engineering and Electronics, 2014, 36 (8): 1660-1667.)

[17] Abdel-Basset M, El-Shahat D, Sangaiah A K. A modified nature inspired meta-heuristic whale optimization algorithm for solving 0–1 knapsack problem [J]. International Journal of Machine Learning & Cybernetics, 2017 (1): 1-20.

[18] Abdel-Basset M, Luo Qifang, Miao Fahui, *et al.* Solving 0–1 knapsack problems by binary dragonfly algorithm [C]// Proc of International Conference on Intelligent Computing. 2017: 491-502.

[19] Cotta C, Troya J M. A hybrid genetic algorithm for the 0-1 multiple knapsack problem [J]. Artificial Neural Nets & Genetic Algorithms, 1998: 250-254.

[20] Hu Zhijun, Li Rong. Ant colony optimization algorithm for the 0-1 knapsack problem based on genetic operators [C]// Advanced Materials Research. 2011: 973-977.

[21] Zhao Jiangfei, Huang Tinglei, Pang Fei, *et al.* Genetic algorithm based on greedy strategy in the 0-1 knapsack problem [C]// Proc of International Conference on Genetic & Evolutionary Computing. 2010: 105-107.

[22] Yang Xinshe. A new metaheuristic bat-inspired algorithm [J]. Computer Knowledge & Technology, 2010, 284: 65-74.

[23] 李枝勇, 马良, 张惠珍. 求解 0//1 背包问题的自适应元胞粒子群算法 [J]. 计算机工程, 2014, 40 (10): 198-203. (Li Zhiyong, Ma Liang, Zhang Huizhen. Adaptive cellular particle swarm algorithm for solving 0//1 knapsack problem [J]. Computer Engineering, 2014, 40 (10): 198-203.)

[24] 吴聪聪, 贺毅朝, 陈嶷瑛, 等. 求解 0-1 背包问题的二进制蝙蝠算法 [J]. 计算机工程与应用, 2015, 51 (19): 71-74. (Wu Congcong, He Yichao, Chen Yiying, *et al.* Binary bat algorithm for solving 0-1 knapsack problem [J]. Computer Engineering and Application, 2015, 51 (19): 71-74.)